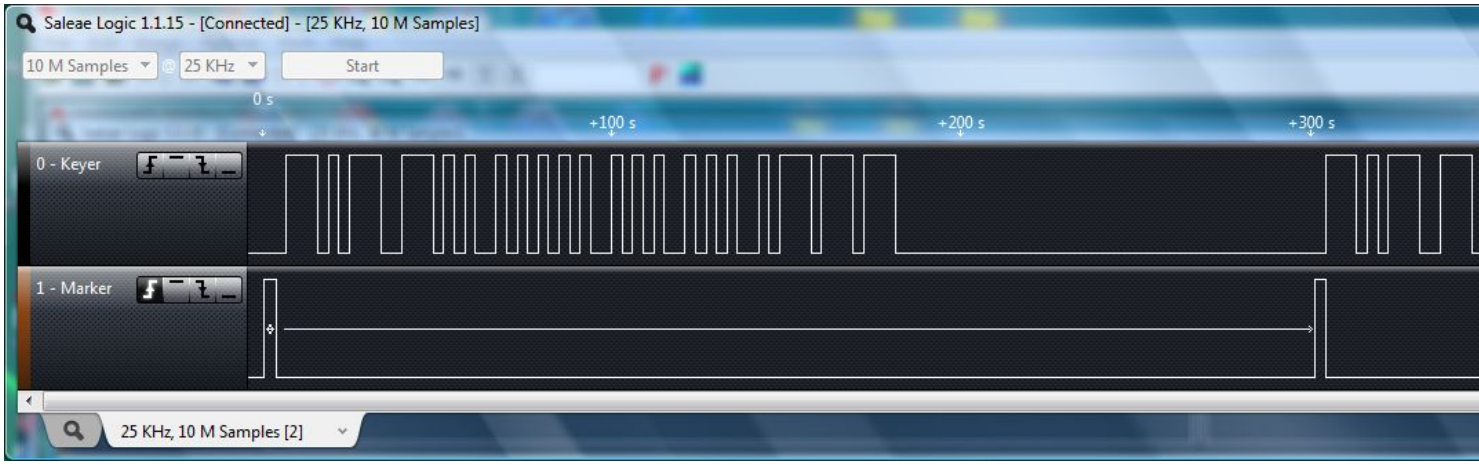
I have completed the first cut on a QRSS Keyer programmed for the Texas Instruments Launch Pad board and programming system. I started with code that Mark VandeWettering K6HX published.



It's all in the timing

Mark's code gave me place to start, but it had one feature I just had to change, the timing was done using a series of delay loops. The QRSS code is sent very slowly so I only needed to be updating it approximately ever 3 or 6 seconds (more later on that), sending of the Morse code will be essentially a background task.

My objection to the delay loop:

- approximately 3 million clock cycles (instructions) are wasted in each loop,
- I wanted to precisely control the timing,
- I wanted to have a precisely repeating cycle for stacking (i.e.10 minutes)
- future expansion plans.

If I rewrote the code to use the hardware timer of the MSP430G2231 (the microprocessor chip) so I could do other things and use an interrupt to give me precice timing intervals.

Why use the Launch Pad

I chose to use the TI Launch Pad because it was just so darn cheap USD $4.30. For that you get a complete microprocessor development system and a platform that is powered and programmed through the USB cable. The Code Composer Studio (CCS) provides a complete software development environment and can be downloaded from TI at no cost. It's a hugh package >>600Mbytes and will take a while to setup on your computer.

MSP430 Microprocessor Chip

The MSP430G2231 (Value Line) processor furnished with the Launch Pad is more than capable for this task. This chip has 2Kbytes of Flash Program memory, 128 bytes of RAM, one timer and Universal Serial Interface (USI), plus much more.

MSP430 Timer with Exteranal XTAL

The feature that I use most is the timer. This chip has a single A2 type timer but it is more than capable. Also supplied with the Launch Pad is a 32,768 Hz crystal but it must be soldered onto the board and it is tiny so it will be a little bit of a challange for some. With the timer clocked by the external xtal, the timer in the upmode and by loading a single constant into the TACCR0 register I was able to generate an interrupt every 3 seconds, just exactly what I needed. This totally eliminates the need for delay loops. There are several additional features of the timer that I am not using at this time.

The c Program

Download the source code (version 1.0). The source code is well commented. First let be explain how a Morse code character is encoded. A single byte is used to define each character, whose length is variable. In other words, not all characters have the same number of symbols. For instants the characters "E" and "T" have a single symbol, dit and dah, respectively. While the numbers and punctuation have 5 and 6 symbols. The characters are encoded so that each symbol is either a 0 for dit or a 1 for dah. Furthermore, the space between each symbol is automatically inserted and is the length of a dit. When a 0 is encoded the key is down for one symbol time and when a 1 is encoded the key is down for 3 symbol times. The symbols are packed into the byte with the first symbol in the least significant bit of the byte, with each additional symbol (if required) in successive bits of more significance. I will use the character "." (period) to hopefully illustrate this more clearly. Find the line in the code:

```
//                  106,          //'.'
```

The number 106 is equal to binary 01101010, the LSB (far right) is a 0 so a dit will be transmitted, the next bit to the left is a 1 so a dah is sent. The next bit is a 0 so a dit is sent, you get the picture. Now here is the trick, the byte is shifted to the right each time the symbol is sent. For example, 01101010 -> 00110101 -> 00011010 -> 00001101 -> 00000110 -> 00000011 -> 00000001 which is the end of the character. The character is finished when the byte has been shifted enough times that the value equals 1 (00000001 binary). To illustrate with a shorter character "E":

```
2,            //'E'
```

The number 2 is equal to binary 00000010. So the LSB is a 0 and we send a dit. After shifting 000000010 -> 00000001 which is equal to 1 and this indicates the character is finished. Pretty clever isn't it, I wish I could take the credit for it but the first time I saw it explained was in a QST magazine probably in the 1990's.

There are several variables defined with global scope and within the main routine are additional variables defined with local scope. There is a structure which defines the elements of the array morsetab[]. There are three short subroutines, dit(void) and dah(void) and the __interrupt void Timer_A (void) and two functions lookup(char) and main(void).

I have used a "state machine" design so that I could overcome the major objection to the original code, see "It's all in the timing" above. Within the the main function we start with a while (1) loop which continues indefinitely and within this loop is a test to determine if the flag tick ==1 has been set. It will be set to 1 each time the timer reaches it's terminal count. Within the conditional test of tick we knock down (reset to 0) the flag and then process all of the things that need to be done for each interrupt. Next we increment the the inteveral variable, which just keeps track of the number of ticks (3 second intervals). Next we test to see if the interval == key_interval, and when it does the next event related to keying must be taken care of. When I

refer to keying I'm using the metaphor of the telegraphers key to send Morse code. When the key is held down it is in short bursts (short is relative) and long bursts (long is relative to a short burst and in this case 3 times longer). The dit subroutine extends the key down event until the next tick, while the dah subroutine extends it for 3 additional ticks. In between symbols we extend the interval for one additional tick with the key up. And between the symbols of the current character and the next character we insert an additional tick (for a total of 2 ticks). Recall that the end of a character is when the value is 1 which is stored in the p variable.

The function lookup is called to get the encoded value for the next character in the string variable. I have used the c language convention of a 0 or nul terminated string. When the end of the string is reached there is nothing else to do. The next_cycle keeps track of how many ticks are needed before we start the cycle again. I am experimenting with a cycle of 5 minutes, this would send my call sign twice in a ten minute window, I'm trying to determine if this gives me a more visually recognizable pattern. You can modify the constants to change this behaviour, as well as, the length of the dit length make it 6 or 10 if you like.

In the future I expect to add the code necessary to initialize a DDS (Direct Digital Synthesizer) for generating the carrier frequency and modulating frequency. This will add band agility and other experimental modes. I'm open for suggestions and comments, please direct them to my email
.